STOL Functions

STOL Functions

This document describes the functions available in ITOS STOL and other applications that use the Sx library.

ABS – Absolute value of x

`ABS(x)` computes the absolute value of *x*. If *x* is float (or string which can be converted to float), the result of `ABS(x)` is float. Otherwise *x* must be int or convertible to int, and the result of *ABS(x)* is int.

```
ABS(15) ⇒ (int) 15
ABS(-3.9) ⇒ (float) 3.9
```

ACOS – Compute arc cosine of x

`ACOS(x)` computes the arc cosine of *x*. *x* must range from -1 to 1; the resulting angle is in radians and will range from 0 to pi.

ASIN – Compute arc sine of x

`ASIN(x)` computes the arc sine of *x*. *x* must range from -1 to 1; the resulting angle is in radians and will range from -pi/2 to pi/2.

ATAN – Compute arc tangent of x

`ATAN(x)` computes the arc tangent of `x`. The resulting angle is in radians and will range from -pi/2 to pi/2.

BWAND – Bitwise AND of x and y

`BWAND(x,y)` computes the bitwise AND of `x` and `y`. `x` and `y` must be unsigned (or convertible to unsigned); the result of `BWAND(x,y)` is unsigned.

```
BWAND(b'00110011',b'10101010') ⇒ b'00100010'
BWAND(123,55) ⇒ 51
```

BWINVERT – Invert bits in x

`BWINVERT(x)` inverts the bits in `x`. `x` must be unsigned (or convertible to unsigned); the result of `BWINVERT(x)` is unsigned.

```
BWINVERT(x'F0F0F0F0') ⇒ (unsigned) x'0F0F0F0F'
```

BWLSHIFT – Logical left shift x y bits

`BWLSHIFT(x,y)` computes the value of *x* shifted left *y* bits. (If *y* is negative, this amounts to a right shift). `y` must be .GT. -32 and .LT. 32. The result of `BWLSHIFT(x,y)` is unsigned.

```
BWLSHIFT(b'0001',3) ⇒ b'1000'
BWLSHIFT(1,3) ⇒ (unsigned) 8
```

BWOR – Bitwise OR of x and y

`BWOR(x,y)` computes the bitwise OR of `x` and `y`. `x` and `y` must be unsigned (or convertible to unsigned); the result of `BWOR(x,y)` is unsigned.

```
BWOR(b'00110011',b'10101010') ⇒ b'10111011'
BWOR(123,55) ⇒ 127
```

BWRSHIFT – Logical right shift x y bits

`BWRSHIFT(x,y)` computes the value of *x* shifted right *y* bits. (If *y* is negative, this amounts to a left shift). *y* must be .GT. -32 and .LT. 32. The result of `BWRSHIFT(x,y)` is unsigned.

```
BWRSHIFT(b'1000',3) ⇒ b'0001'
BWRSHIFT(100,3) ⇒ (unsigned) 12
```

CEIL – Smallest int .GE. x

`CEIL(x)` computes the smallest int that is .GE. `x`. `x` must be float (or convertible to float); the result of `CEIL(x)` is int.

```
CEIL(4.8) ⇒ (int) 5
CEIL(-4.8) ⇒ (int) -4
```

CONCAT – Concatenate strings (2 to 999)

`CONCAT(x,y,...)` concatenates all arguments into a string. The result of `CONCAT(x,y,...)` is a string.

```
CONCAT("name",1) ⇒ "name1"
```

To sequential print to a file whose name is generated from the year and day of year, do something like:

```
GLOBAL day,year
day = 100
year = 93
SEQPRT acstmp > (concat("acstmp.",year,"-",day))
```

The above example sequential prints to `acstmp.93-100`.

COS – Cosine of x

`COS(x)` computes the cosine of `x`. `x` must be float (or convertible to float); the result of `cos(x)` is float. `x` is in radians.

```
COS(0) ⇒ (float) 1
COS(.123) ⇒ (float) 0.992445
```

DELTA

This function is not yet implemented. It is intended to be used with `WAIT UNTIL` to determine the delta difference between consecutive values (to make it possible to 'wait until the value changes by more than 1.7', for example).

EVTMSG

This function is not yet implemented. It is intended to be used for it's side effect, which is to generate an event message.

EXISTS – Has mnem's value been set?

`EXISTS(`*mnemonic*`)` returns 1 if *mnemonic* is, in fact, a mnemonic and *mnemonic*'s value exists.

A mnemonic's value is said to exist if it has ever been set. Most global mnemonics get set (and thus exist) when the system starts up. Telemetry mnemonics don't exist until they've been received in telemetry (see ⟨undefined⟩ [Exist flag], page ⟨undefined⟩).

FLOOR – Largest int .LE. x

`FLOOR(x)` computes the largest int that is .LE. `x`. `x` must be float (or convertible to float); the result of `FLOOR(x)` is int.

```
FLOOR(4.8) ⇒ (int) 4
FLOOR(-4.8) ⇒ (int) -5
```

FORMAT – Format x

The `FORMAT(fmt,x)` function creates a string containing the value of `x` formatted according to `fmt`.

`fmt` is similar to a C language printf conversion specification and is a string beginning with the % character, ending with the conversion character, and possible containing conversion parameters in between. Note that `format` must begin with the % character and must end with the conversion character!

The optional conversion parameters are the conversion flags, minimum field width, and precision; an example of a conversion with all of the optional conversion parameters is `%+8.4f` – `+` is a conversion flag; `8` is the minimum field width, and `4` is the precision.

`diouxXb` The int (or appropriate variant) argument is converted to signed decimal (`d` and `i`), unsigned octal (`o`), unsigned decimal (`u`), unsigned hexadecimal (`x` and `X`), or unsigned binary (`b`) notation. The letters '`abcdef`' are used for `x` conversions; the letters '`ABCDEF`' are used for `X` conversions. The precision, if any, gives the minimum number of digits that must appear; if the converted value requires fewer digits, it is padded on the left with zeros.

`eE`	The double argument is rounded and converted in the style [-]d.ddde+-dd where there is one digit before the decimal-point character and the number of digits after it is equal to the precision; if the precision is missing, it is taken as 6; if the precision is zero, no decimal-point character appears. An E conversion uses the letter `E` (rather than `e`) to introduce the exponent. The exponent always contains at least two digits; if the value is zero, the exponent is 00.
`f`	The double argument is rounded and converted to decimal notation in the style [-]ddd.ddd, where the number of digits after the decimal-point character is equal to the precision specification. If the precision is missing, it is taken as 6; if the precision is explicitly zero, no decimal-point character appears. If a decimal point appears, at least one digit appears before it.
`g`	The double argument is converted in style f or e (or E for G conversions). The precision specifies the number of significant digits. If the precision is missing, 6 digits are given; if the precision is zero, it is treated as 1. Style e is used if the exponent from its conversion is less than -4 or greater than or equal to the precision. Trailing zeros are removed from the fractional part of the result; a decimal point appears only if it is followed by at least one digit.
`c`	The int argument is converted to an unsigned char, and the resulting character is written.
`s`	The argument is converted to a character string. Characters are written up to (but not including) a terminating NUL character. If a precision is specified, no more than the number specified are written.

Some format examples:

```
FORMAT("%b",22)
⇒ 10110
FORMAT("%08b,22)
⇒ 00010110
FORMAT("%08X,22)
⇒ 00000016
FORMAT("%s","abcd")
⇒ abcd
FORMAT("%12s","abcd")
⇒         abcd
```

GETENV – get environment variable

The `GETENV` function returns the value of an environment variable.

```
SHOVAL GETENV(SHELL)
⇒ "/bin/tcsh"
```

ISCOMMAND – Is name a command mnemonic?

`ISCOMMAND(name)` tests whether or not `name` is a command mnemonic in the database. The result of `ISCOMMAND(name)` is non-zero if `name` is a command mnemonic. For cases that do not map the command database such as STOL, this call will always return 0.

ISDATE – Is x a date?

`ISDATE(x)` returns int 1 iff `x` evaluates to a date, and int 0 otherwise.

ISFLOAT – Is x a float?

`ISFLOAT(x)` returns int 1 iff `x` evaluates to a float, and int 0 otherwise.

```
ISFLOAT(1.2) ⇒ (int) 1
ISFLOAT(1) ⇒ (int) 0
```

ISGLOBAL – Is name a global variable?

`ISGLOBAL(name)` tests whether or not `name` is a STOL global variable. (ISGLOBAL does _not_ test for global mnemonics).

ISINLIMITS – Is the mnemonic in limits?

`ISINLIMITS(name)` tests whether or not `name` is a mnemonic within limits. `ISINLIMITS(name)` returns int 1 iff `name` is in limits, and int 0 otherwise.

ISINT – Is x an int?

`ISINT(x)` returns int 1 iff `x` evaluates to an int, and int 0 otherwise.

```
ISINT(1.2) ⇒ (int) 0
ISINT(1) ⇒ (int) 1
```

ISLOCAL – Is name a local variable?

`ISLOCAL(name)` tests whether or not `name` is a STOL local variable. `ISLOCAL(name)` returns int 1 iff `x` is a STOL local variable, and int 0 otherwise.

ISMNEMONIC – Is name a telemetry mnemonic?

`ISMNEMONIC(name)` tests whether or not `name` is a telemtry mnemonic in the database. The result of `ISMNEMONIC(name)` is non-zero if `name` is a telemetry mnemonic. For cases that do map the telemetry database, this call will always return 0.

ISNULL – Is x null?

`ISNULL(x)` returns int 1 iff `x` is unset (i.e. NULL), and int 0 otherwise.

ISNUMBER – Is x an int, unsigned or float?

`ISNUMBER(x)` returns int 1 iff `x` is number, and int 0 otherwise.

ISREDHI – Is mnem's value red high?

`ISREDHI(name)` returns 1 iff `name` mnemonic is in red high limit violation, and int 0 otherwise.

ISREDLO – Is mnem's value red low?

`ISREDLO(name)` returns 1 iff `name` mnemonic is in red low limit violation, and int 0 otherwise.

ISSTATIC – Has mnem gone static?

`ISSTATIC(mnemonic)` returns 1 if `mnemonic` is static and returns 0 otherwise.

A mnemonic goes static if it isn't received in telemetry within an appropriate interval of time (see ⟨undefined⟩ [Static flag], page ⟨undefined⟩).

ISSTRING – Is x a string?

`ISSTRING(x)` returns int 1 iff `x` evaluates to a string, and int 0 otherwise.

ISSYMBOL – Is name a variable or mnemonic?

`ISSYMBOL(x)` returns int 1 iff `x` is a STOL variable or database mnemonic, and int 0 otherwise.

ISTIME – Is x a time?

`ISTIME(x)` returns 1 iff `x` evaluates to a time, and int 0 otherwise.

ISUNSIGNED – Is x unsigned?

`ISUNSIGNED(x)` returns 1 iff `x` evaluates to unsigned, and int 0 otherwise.

```
ISUNSIGNED(0) ⇒ (int) 0
ISUNSIGNED(x'00') ⇒ (int) 1
```

ISVARIABLE – Is name a variable?

`ISVARIABLE(name)` returns int 1 iff `name` is a STOL variable, and int 0 otherwise.

ISYELLOWHI – Is mnem's value yellow high?

`ISYELLOWHI(name)` returns 1 iff `name` mnemonic is in yellow high limit violation, and int 0 otherwise.

ISYELLOWLO – Is mnem's value yellow low?

`ISYELLOWLO(name)` returns 1 iff `name` mnemonic is in yellow low limit violation, and int 0 otherwise.

LN – Natural log of x

`LN(x)` computes the natural log of `x`. `x` must be greater than 0.

LOG – Base 10 log of x

`LOG(x)` computes the base 10 log of `x`. `x` must be greater than 0.

LOWERCASE – Convert string to lower case

`LOWERCASE(x)` returns the lowercase equivalent of `x`.

MAX – Find largest number in list of 1 to 999 items

`MAX(x,y,...)` returns the largest number in the argument list. Works with both integers and floats or a mixture of both.

MIN – Find smallest number in list of 1 to 999 items

`MIN(x,y,...)` returns the smallest number in the argument list. Works with both integers and floats or a mixture of both.

MKDATE – Make date from raw secs and subsecs

The `MKDATE` function constructs a date from raw seconds and subseconds.

```
MKDATE(808003397,1234) ⇒ 93-364-21:23:17.018829
```

MKEPOCHDATE – Make date since given epoch from raw secs and subsecs

The `MKEPOCHDATE(epoch, sec, subseconds)` function constructs a date from raw seconds and subseconds based on given epoch.

```
MKEPOCHDATE(GBL_DEF_EPOCH,808003397,1234) ⇒ 93-364-21:23:17.084364

MKEPOCHDATE(68-145-00:00:00.065535,808003397,1234) ⇒ 93-364-21:23:17.084364
```

MKTIME – Make time from raw secs and subsecs

The `MKTIME(sec, subseconds)` function constructs a time from raw seconds and subseconds.

```
SHOVAL MKTIME(123,456) ⇒ 0:02:03.006958
```

MOD – Find remainder when x / y

The `MOD(x,y)` function can be used to determine if a number is odd or even:

```
MOD(112,2) ⇒ 0
MOD(113,2) ⇒ 1
MOD(114,2) ⇒ 0
```

NAME – Use string as a name

Allows the name of a command or submnemonic to be passed as an argument to a proc, as in:

```
PROC ABC(S1,S2)
/SOMECMD NAME(S1)=1, NAME(S2)=1
ENDPROC
```

which, when started with `START ABC(XYZ,PDQ)`, issues the command `/SOMECMD XYZ=1, PDQ=1`. Whew. Another, less useful, example might make things more clear:

```
LOCAL X,Y
Y = 543
X = "Y"
SHOVAL NAME(X)
⇒ global Y: (int) 543
```

SIN – Sine of x

`SIN(x)` computes the sine of `x`. `x` must be float (or convertible to float); the result of `SIN(x)` is float. `x` is in radians.

SQRT – Square root of x

`SQRT(x)` computes the square root of `x`. `x` must be float (or convertible to float); the result of `SQRT(x)` is float.

STRFDATE – Format date

`STRFDATE(fmt,date)` creates a string containing the value of `date` formatted according to `fmt`.

`fmt` is a format string as used by the C library function strftime(). From `man 3 strftime`:

`Aa`	is replaced by the weekday name. `A` produces the full name; `a` produces the abbreviated name.
`Bbh`	is replaced by the month name. `B` produces the full name; `b` and `h` produce the abbreviated name.
`C`	shorthand for '`%a %b %e %H:%M:%S %Y`'.
`c`	shorthand for '`%m/%d/%y %H:%M:%S`'.

`D`	shorthand for '`%m/%d/%y`'.
`de`	is replaced by the 2-digit day of the month, 1 to 31. `d` preceeds single digits with '`0`'; `e` with space.
`f`	is replaced by the microseconds field of the input time as a 6-digit, zero-filled decimal number.
`Hk`	is replaced by the hour on a 24-hour scale, 0-23. `H` preceeds single digits with '`0`'; codek with space.
`Il`	is replaced by the hour on a 12-hour scale, 1-12. `I` preceeds single digits with '`0`'; `l` with space.
`Jj`	is replaced by the day of the year (Julian day). `J` begins with day '`000`'; `j` with day '`001`'.
`M`	is replaced by the minute, '`00`'-'`59`'.
`m`	is replaced by the month, '`01`'-'`12`'.
`n`	is replaced by a newline, which cannot appear on a page, but might be useful to an ITOS programmer.
`p`	is replaced by '`AM`' or '`PM`', as appropriate.
`R`	is shorthand for '`%H:%M`'.
`r`	is shorthand for '`%I:%M:%S %p`'.
`S`	is replaced by the number of seconds, '`00`'-'`59`'.
`s`	is replaced by the number of seconds in the input time, which are seconds from the UNIX epoch in local time.
`TX`	is shorthand for '`%H:%M:%S`'.
`t`	is replaced by a tab character, which is not useful in page definitions.
`UW`	is replaced by the week number of the year, '`00`'-'`53`'. `U` takes Sunday as the first day of the week; `W` takes Monday.
`w`	is replaced by the day of the week, '`0`'-'`6`', with Sunday as day '`0`'.
`x`	is shorthand for '`%m/%d/%y`'.
`Yy`	is replaced by the year. `y` produces the 2-digit year, '`00`'-'`99`'; '`Y`' the four-digit year.
`Z`	is replaced by the timezone abbreviation.

The following examples use `STRFDATE` to create log file names:

```
LOG >> (CONCAT("LOG.",STRFDATE("%m-%d",P@GBL_GMTOFF)))
⇒LOG.06-20
LOG >> (CONCAT("LOG.",STRFDATE("%b%d",P@GBL_GMTOFF)))
⇒LOG.Jun20
```

STRLEN – Determine length of string.

`STRLEN(x)` returns the number of characters in string `x`. `x` must be a string (or convertible to string); the result of `STRLEN(x)` is int.

```
STRLEN("123") ⇒ (int) 3
```

STRTOL – Convert a string to signed int

`STRTOL(string,b)`[1] returns the number in base `b` contained in `string`.

```
STRTOL("FF",16) ⇒ 255
STRTOL("10110",2) ⇒ 22
STRTOL("10110junk",2) ⇒ 22
STRTOL("junk",2) ⇒ (Operator error)
```

If `base` is not specified, 10 is assumed.

`STRTOL` can be used in conjuction with the `ASK` directive to allow the Test Conductor to enter a hex number:

```
LOCAL hex,decimal
ASK "enter a hex number", hex
decimal = STRTOL(hex,16)
shoval decimal
```

In the above example, if the Test Conductor enters `16` in response to the `ASK` directive's prompt, the `SHOVAL` directive's result is 22!

STRTOUL – Convert a string to unsigned int

`STRTOUL(string,b)`[2] is similar to STRTOL except it returns an unsigned number.

```
sho strtoul("0xf6741123",16)
⇒ 4134801699
sho strtol("0xf6741123",16)
⇒ 2147483647
```

SUBSTR – Extract substring

`SUBSTR(x,y,z)` extracts a substring from string `x`. Int `y` is the index of the first character in the substring (the first character in `x` has index 1). Optional int `z` is the number of characters in the substring; if `z` isn't specified, the substring uses the remaining characters in `x`. `x`, `y`, and `z` may be expressions.

```
SUBSTR("abcd",2) ⇒ "bcd"
SUBSTR("93-344-19:40:33",4,3) ⇒ "344"
```

The following directive logs event messages to a file named `LOG-ddd` where `ddd` is the current day-of-year:

```
LOG > (CONCAT("LOG-", SUBSTR(P@GBL_GMTOFF,4,3)))
```

[1] It's named STRTOL because it's modeled after the C library function strtol()

[2] It's named STRTOUL because it's modeled after the C library function strtoul()

TAN – Tangent of x

`TAN(x)` computes the tangent of `x`. `x` must be float (or convertible to float); the result of `TAN(x)` is float. `x` is in radians.

TODATE – Convert x to date

`TODATE(x)` converts `x` to a date. When converting and integral, float, or time value to date the value is treated as the number of seconds since the midnight that began Jan 1 1970.

TOFLOAT – Convert x to float

`TOFLOAT(x)` returns the float equivalent of `x`.

TOINT – Convert x to int

`TOINT(x)` returns the int equivalent of `x`.

TOSTRING – Convert x to string

`TOSTRING(x)` returns the string equivalent of `x`.

TOTIME – Convert x to time

`TOTIME(x)` returns the time equivalent of `x`.

TOUNSIGNED – Convert x to unsigned

`TOUNSIGNED(x)` returns the unsigned equivalent of `x`.

UPPERCASE – Convert string to uppercase

`UPPERCASE(x)` returns the uppercase equivalent of `x`.

Table of Contents